Final Automne - 20 Janvier 2006
Durée : 2 heures (10h30 - 12h30)
Aucun document autorisé

Bases Fondamentales de la P.O.O. (LO43)

1 Questions de Cours

1.1 Une question d'amitié

En C++, il est possible de définir des méthodes ou des classes "Amies" pour accèder à des champs privés inaccessibles. Ce mécanimse permet d'éviter de définir une relation d'héritage entre les classes concernées ou de changer le niveau de protection des données (passage de privé à public par exemple ce qui est catastrophique d'un point de vue conception). Ce mécanisme d'amitié n'existe pas en Java, pourquoi? Quelle notion le remplace?

1.2 Eléments de généricité

Jusqu'à présent Java ne disposait pas de mécanismes analogues aux template(s) en C++. C'est maintenant chose faite avec les generics de la version 1.5. Expliquez les différences entre les generics et template du C++ (syntaxe, mécanismes, possibilités,...). Vous ecrirez un petit programme permettant de mettre en évidence le fait que, contrairement au C++, il n'y a pas en Java de duplication de code compilé en fonction de la spécification du paramère formel T.

2 Exercice : Passage de paramètres

On veut écrire une méthode permettant de permuter ("swapper") 2 objets.

```
 void swap (Object o1, Object o2) {
Object temp;
```

```
temp = o1;
o1 = o2;
o2 = temp;}
```

Le code précédent ne donne pas le résultat désiré, pourquoi? (vous pouvez vous aider d'un schéma mémoire par exemple)

Proposez une solution pour écrire une méthode de swap qui fonctionne correctement.

3 Exercice : les Exceptions

Qu'affiche le code suivant? Expliquez vos réponses.

```
class MonException extends Exception {
MonException() { super() ; }
MonException(String s) { super(s) ; } }

class MaFaute extends MonException {
MaFaute() { super() ; }
MaFaute(String s) { super(s) ;} }

class MonErreur extends MonException {
MonErreur() { super() ; }
MonErreur(String s) { super(s) ; } }

public class MaClasse {
static int uneMethode (int i) throws Exception {
try {
if (i==0) throw new Exception("Exception") ;
if (i==1) throw new MonException("MonException") ;
if (i==2) throw new MonErreur("MonErreur") ;
if (i==3) throw new MaFaute("MaFaute");

} catch (MonErreur e) {
System.out.println ("uneMethode :  "+e.getMessage()) ;

} finally {
System.out.println ("Finally uneMethode "+i) ;
return 0 ;
}
```

```
}

public static void main(String[] args) {
for (int i = 0 ; i<6;i++) {
try {
uneMethode (i) ;
System.out.println ("Rien ") ;
} catch (MonErreur e) {
System.out.println ("MonErreur :  "+e.getMessage()) ;
} catch (MonException e) {
System.out.println ("MonException :  "+e.getMessage()) ;
} catch (MaFaute e) {
System.out.println ("MaFaute :  "+e.getMessage()) ;
} catch (Exception e) {
System.out.println ("Exception :  "+e.getMessage()) ;
}
}
}
}
```

4 Problème : Conception et implémentation d'une simulation de chaîne de montage

Le but de ce problème est d'effectuer la conception et une partie de l'implémentation d'un outil de simulation d'une chaîne de montage.

4.1 Conception : Simulation d'une chaine de montage simple

On veut simuler une chaîne de montage dans une usine d'assemblage. Une chaîne est constituée de N postes reliés deux à deux par un espace de stockage. Cet espace correspond à la sortie des pièces construites par le poste de rang N et aux ressources nécessaires pour le montage sur le poste de rang N+1. Cet espace permet donc de mettre en attente les produits qui sortent d'un poste et de les acheminer vers le poste suivant dans la chaîne. Chaque poste P consomme de la matière première pour construire un produit à partir du produit semi fini fourni par le poste précédent. Le premier poste de la chaîne de montage n'utilise que de la matière première. Le dernier poste fourni un produit fini qui est stocké dans un hangar. Le fonctionnement de chaque poste possède une contrainte de temps t (différente pour chaque poste) qui correspond au temps nécessaire pour le traitement. Pendant ce temps, le poste est

dans l'état "en fonction". S'il n'y a plus de matière première ou si l'espace de stockage inter-poste est vide, le poste est dans l'état "en attente". Le poste peut également être "en maintenance" en fin de cycle si nécessaire. Le temps de maintenance est fourni par une méthode "qualité" appelée en chaque fin de cycle de fonctionnement. Si le temps fourni par cette méthode est égal à 0 le poste continue le cycle de fonctionnement classique. Le ré-approvisionnement en matières premières est fait tous les temps T (avec éventuellement un décalage aléatoire en fonction de la ponctualité des fournisseurs)

En vous aidant de tous les diagrammes UML que vous jugerez nécessaires, proposez une conception pour cet outil de simulation. Vous metterez en évidence les aspects statiques **et** les aspects dynamiques.

4.2 Implémentation

On choisit de définir un thread pour chacun des postes de montage.

4.2.1 Thread et Java

Quelles sont les deux méthodes pour écrire un Thread en Java? Vous expliquerez les avantages de l'une et de l'autre.

4.2.2 Poste de montage

Donnez l'implémentation de la classe définissant un poste de montage. Vous pouvez utiliser la méthode de votre choix.

4.2.3 Enchainement de poste de montage

Deux postes de montages successifs partagent un espace de stockage. En vous aidant de ce qui a été fait en TD pour le problème du producteur-consommateur, explicitez l'implémentation du fonctionnement de l'échange de pièces entre deux postes successifs.